



307

**FIGURE 1B**

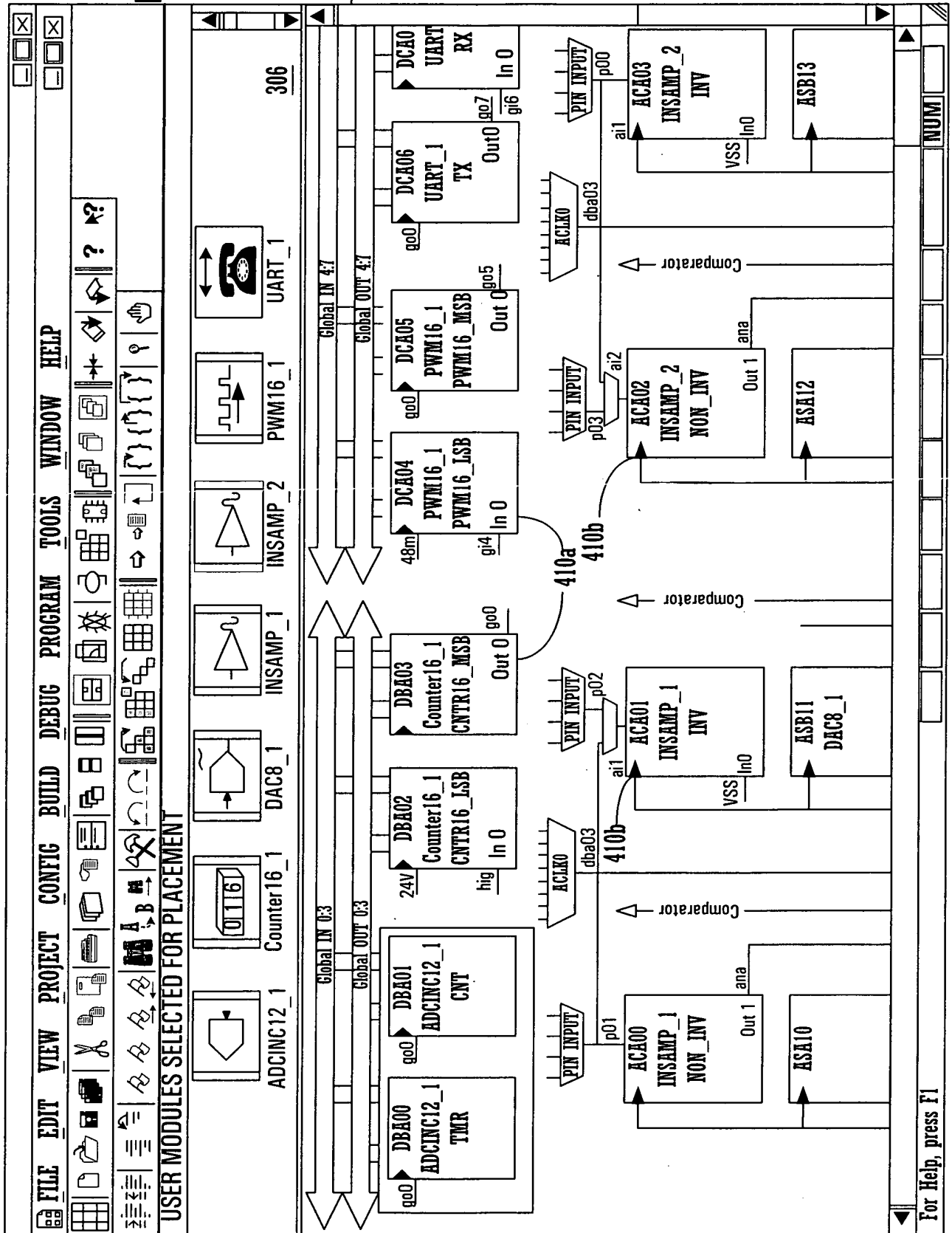
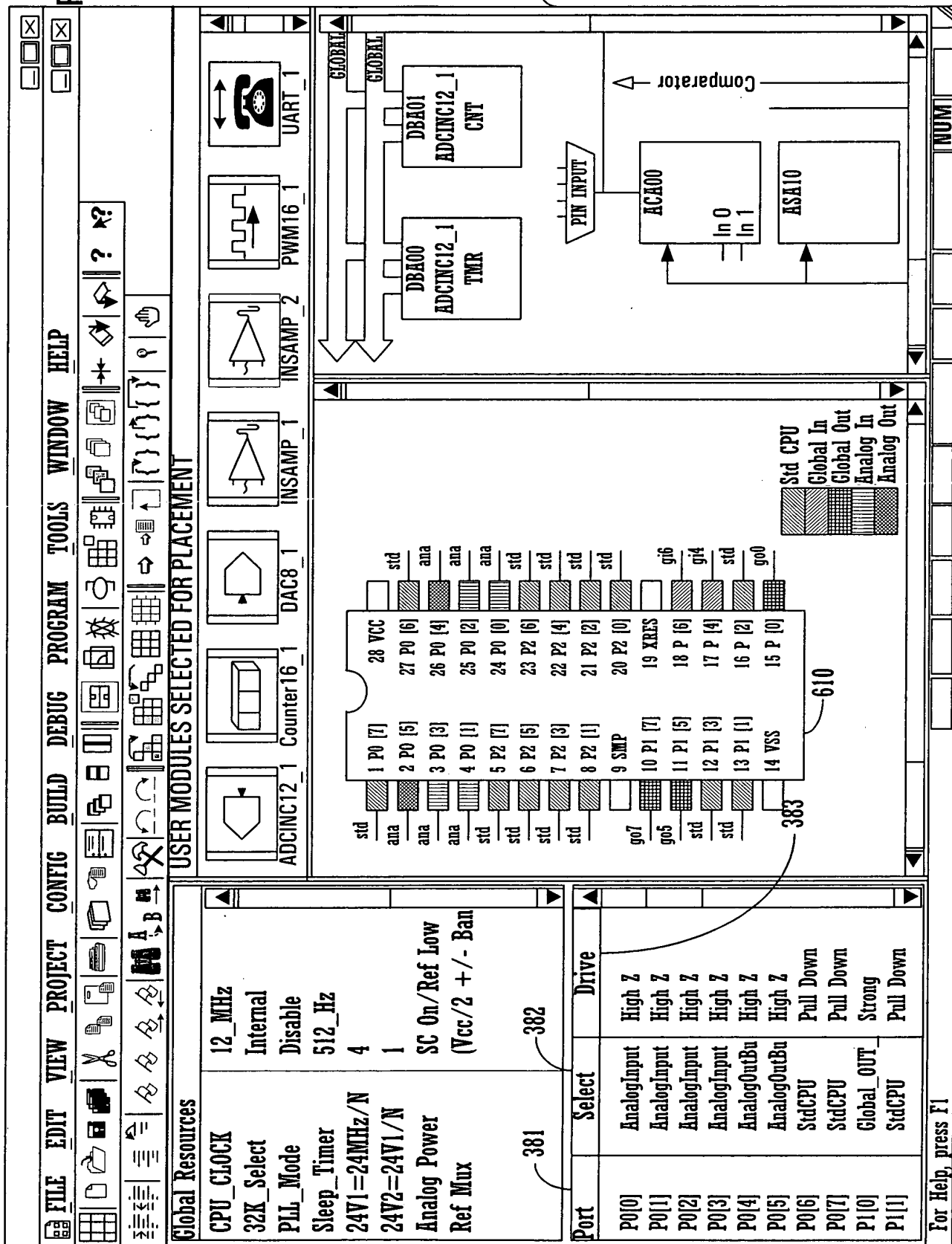


FIGURE  
1C



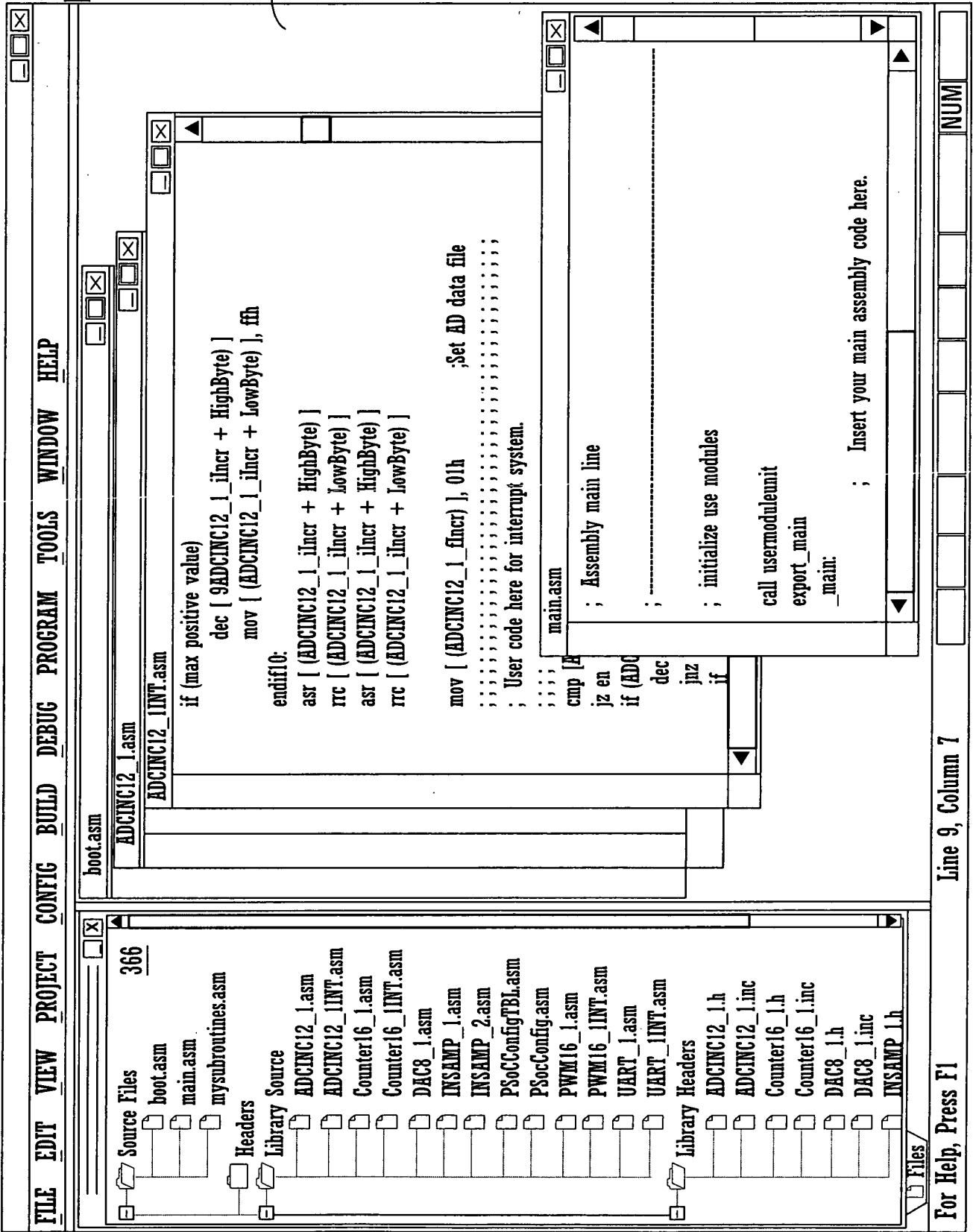


FIGURE 1D

365

5/34

Begin

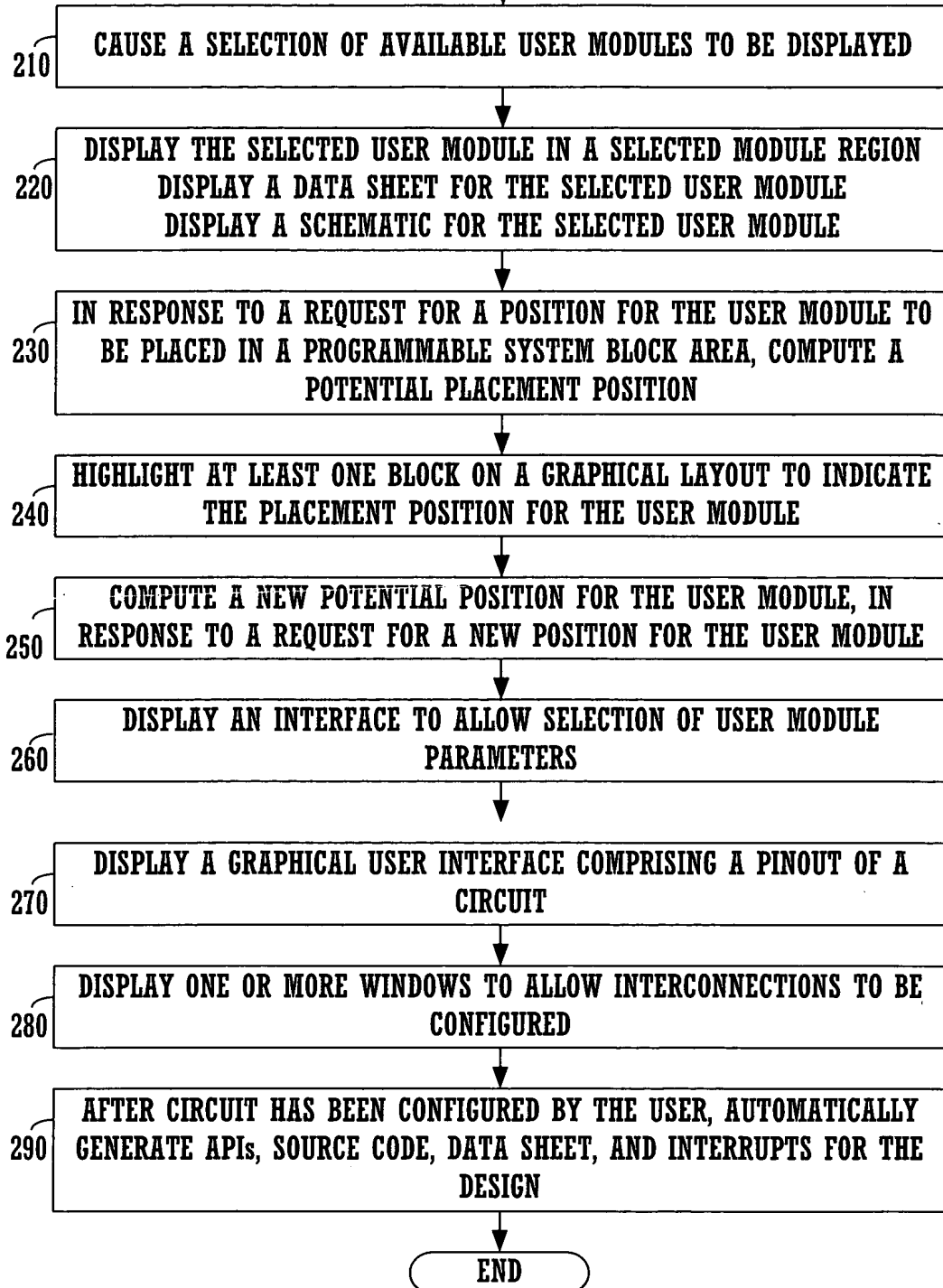


FIGURE 2



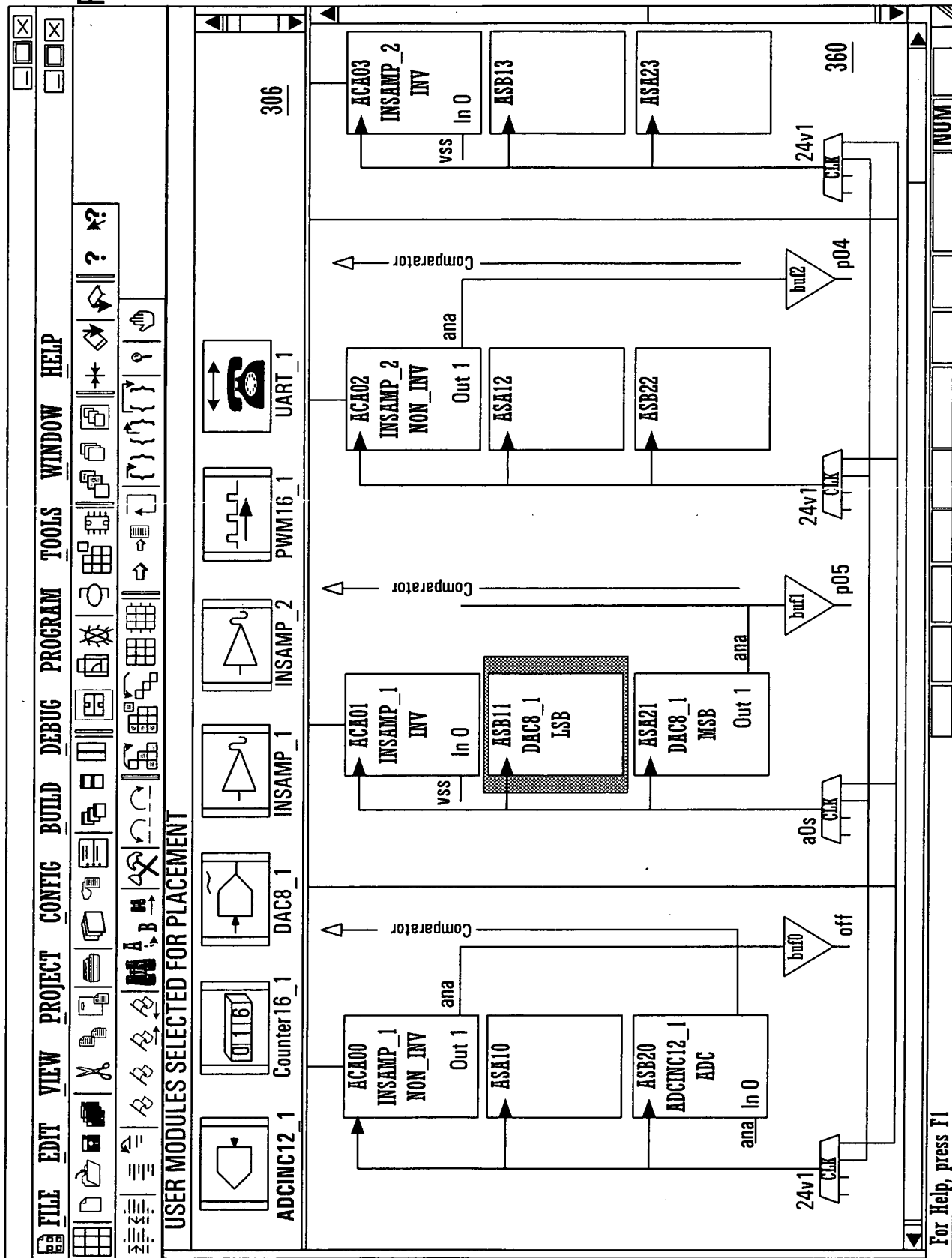
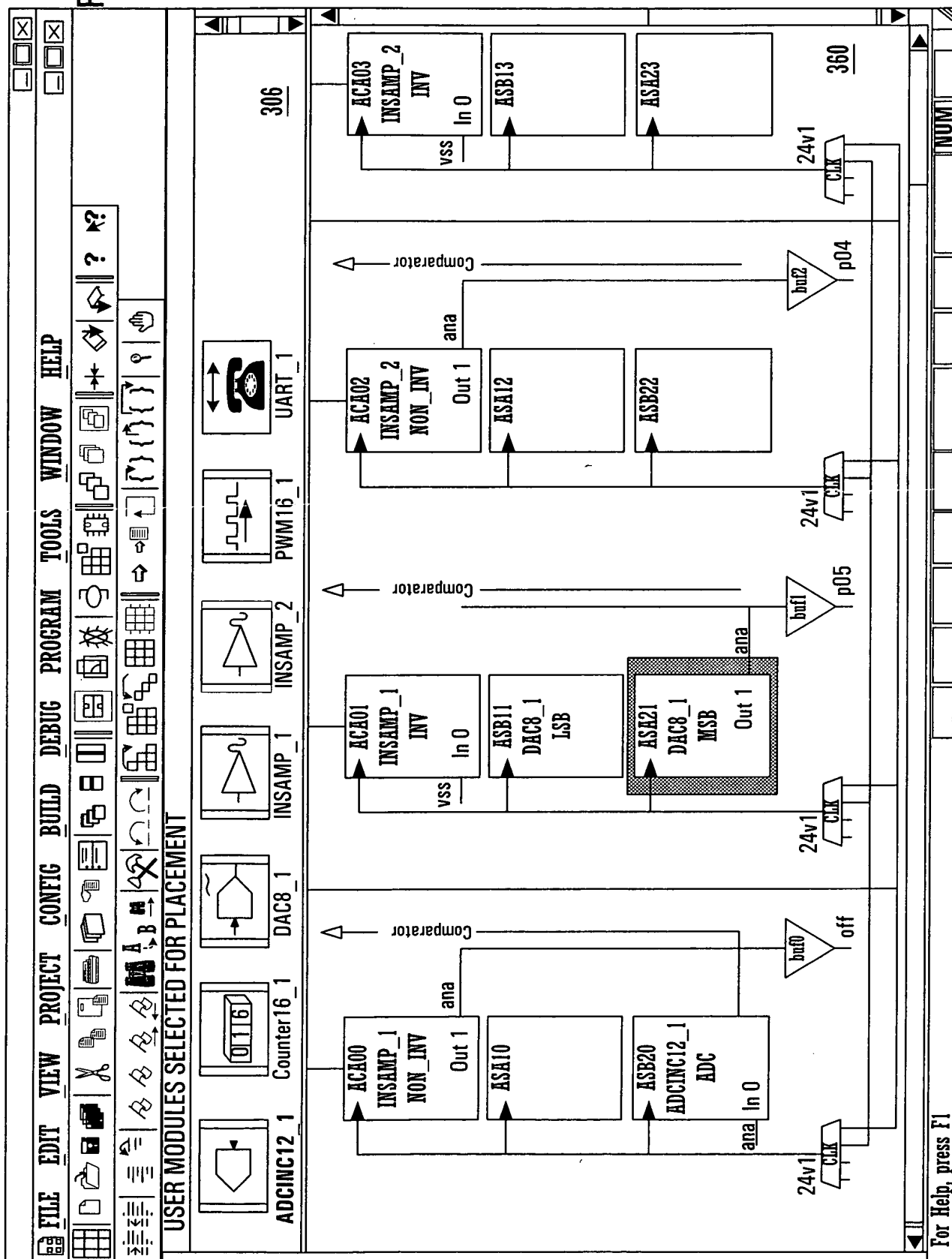


FIGURE  
3B

**FIGURE 3C**





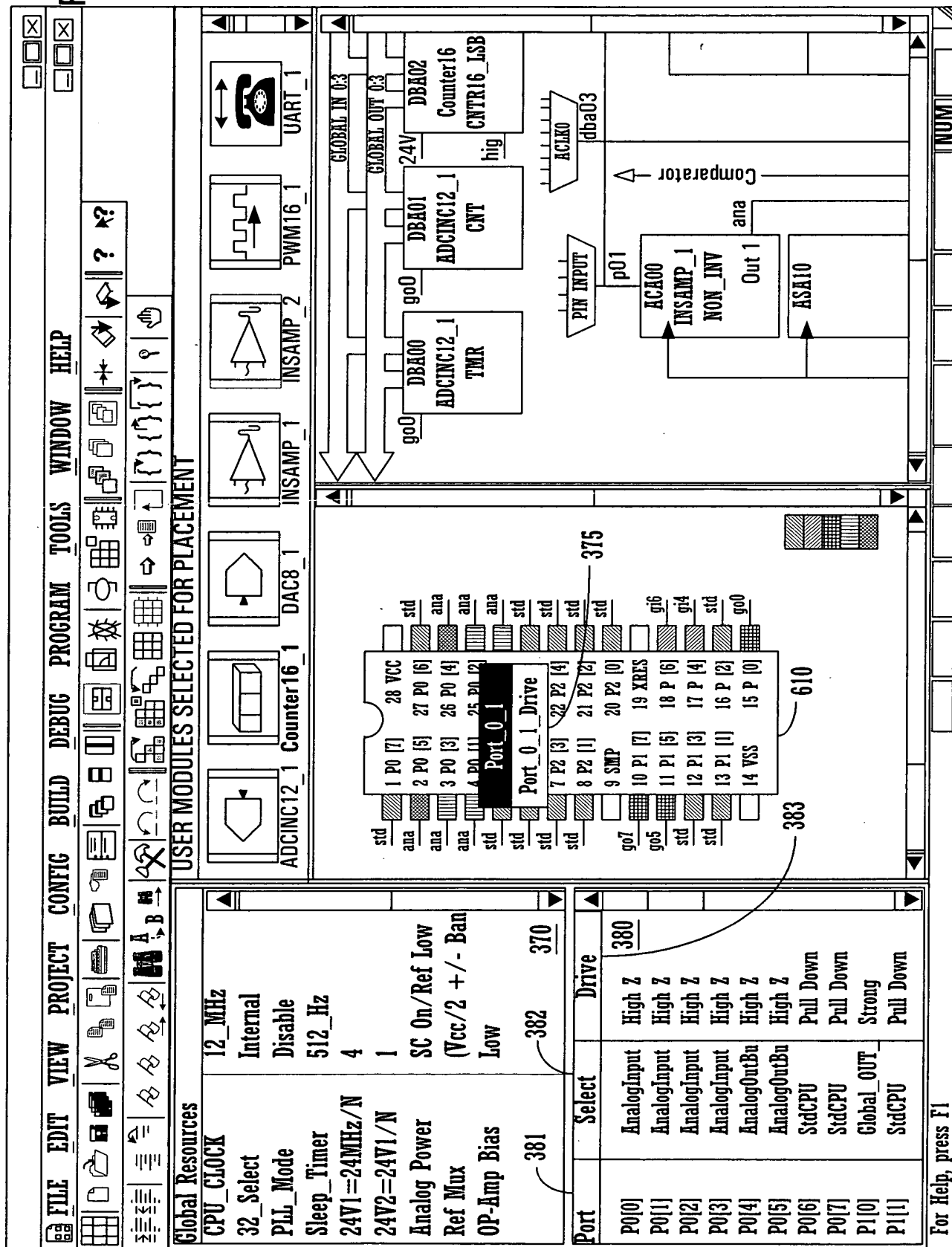
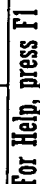


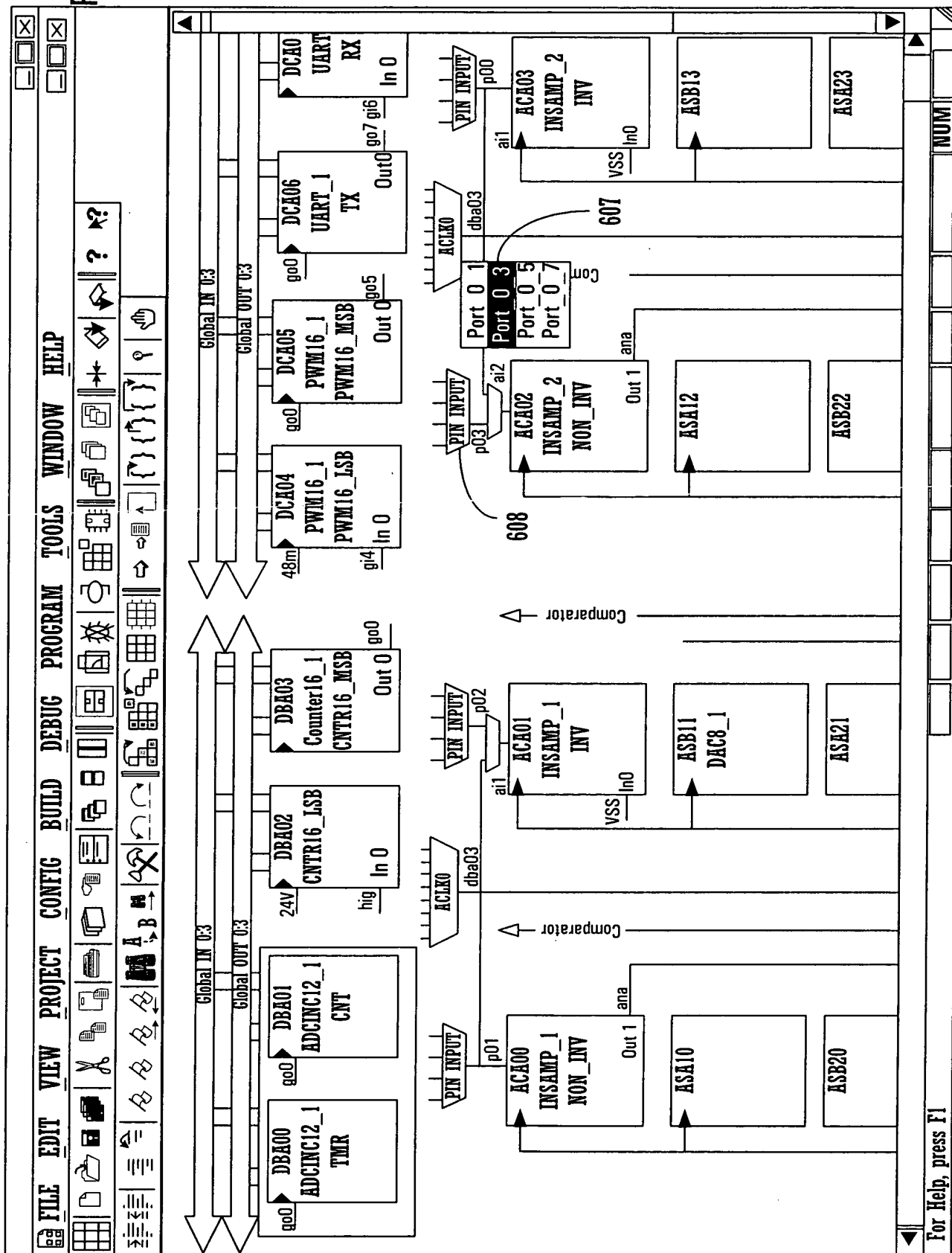
FIGURE  
5A

**FIGURE 5B**

**FIGURE 5C**









configtbl.asm

17/34

; Personalization tables

export LoadConfigTBL\_project\_Bank1

export LoadConfigTBL\_project\_Bank0

LoadConfigTBL\_project\_Bank1:

; Global Register values

db	61h, 03h	;AnalogClockSelect register
db	60h, 08h	;AnalogColumnClockSelect register
db	62h, 30h	;Analog0Control register
db	63h, 00h	;AnalogModulatorControl register
db	e1h, 30h	;OscillatorControl_1 register
db	00h, 00h	;Port_0_DriveMode_0 register
db	01h, 3fh	;Port_0_DriveMode_1 register
db	04h, a1h	;Port_1_DriveMode_0 register
db	05h, 50h	;Port_1_DriveMode_1 register
db	08h, 00h	;Port_2_DriveMode_0 register
db	09h, 00h	;Port_2_DriveMode_1 register
db	0ch, 00h	;Port_3_DriveMode_0 register
db	0dh, 00h	;Port_3_DriveMode_1 register
db	10h, 00h	;Port_4_DriveMode_0 register
db	11h, 00h	;Port_4_DriveMode_1 register
db	14h, 00h	;Port_5_DriveMode_0 register
db	15h, 00h	;Port_5_DriveMode_1 register
db	e3h, 84h	;VoltageMonitorControl register

; Instance name ADCINC12\_1, User Module ADCINC12

; Instance name ADCINC12\_1, Block Name ADC(ASB20)

db	90h, 90h	;ADCINC12_1_AtoDcr0
db	91h, 60h	;ADCINC12_1_AtoDcr1
db	92h, 60h	;ADCINC12_1_AtoDcr2
db	93h, f0h	;ADCINC12_1_AtoDcr3

; Instance Name ADCINC12\_1, Block Name CNT(DBA01)

db	24h, 21h	;ADCINC12_1_CounterFN
db	25h, 48h	;ADCINC12_1_CounterSL
db	26h, 00h	;ADCINC12_1_CounterOS

; Instance Name ADCINC12\_1, Block Name TMR(DBA00)

db	20h, 20h	;ADCINC12_1_TimerFN
db	21h, 18h	;ADCINC12_1_TimerSL
db	22h, 00h	;ADCINC12_1_TimerOS

; Instance name Counter16\_1, User Module Counter 16

; Instance name Counter16\_1, Block Name CNTR16\_LSB(DBA02)

db	28h, 01h	;Counter16_1_FUNC_LSB_REG
db	29h, 16h	;Counter16_1_INPUT_LSB_REG
db	2ah, 00h	;Counter16_1_OUTPUT_LSB_REG

; Instance name Counter16\_1, Block Name CNTR16\_MSB(DBA03)

db	2ch, 21h	;Counter16_1_FUNC_MSB_REG
db	2dh, 36h	;Counter16_1_INPUT_MSB_REG
db	2eh, 04h	;Counter16_1_OUTPUT_MSB_REG

**FIGURE 7A**



```
; Instance name DAC8_1, User Module DAC8
;   Instance name DAC8_1, Block Name LSB(ASB11)
      db          84h, 80h      ;DAC8_1_LSB_CR0
      db          85h, 80h      ;DAC8_1_LSB_CR1
      db          86h, 20h      ;DAC8_1_LSB_CR2
      db          87h, 30h      ;DAC8_1_LSB_CR3
;   Instance name DAC8_1, Block Name MSB(ASA21)
      db          94h, a0h      ;DAC8_1_MSB_CR0
```

**FIGURE 7A (Continued)**

```

db          95h, 41h      ;DAC8_1_MSB_CR1
db          96h, a0h      ;DAC8_1_MSB_CR2
db          97h, 30h      ;DAC8_1_MSB_CR3
; Instance name INSAMP_1, User Module INSAMP
;   Instance name INSAMP_1, Block Name INV(ACA01)
db          75h, beh      ;INSAMP_1_INV_CRO
db          76h, 21h      ;INSAMP_1_INV_CR1
db          77h, 20h      ;INSAMP_1_INV_CR2
;   Instance name INSAMP_2, Block Name NON-INV(ACA00)
db          71h, 3ch      ;INSAMP_1_NON_INV_CRO
db          72h, a1h      ;INSAMP_1_NON_INV_CR1
db          73h, 20h      ;INSAMP_1_NON_INV_CR2
; Instance name INSAMP_2, User Module INSAMP
;   Instance name INSAMP_2, Block Name INV(ACA03)
db          7dh, ceh      ;INSAMP_2_INV_CRO
db          7eh, 21h      ;INSAMP_2_INV_CR1
db          7fh, 20h      ;INSAMP_2_INV_CR2
;   Instance name INSAMP_2, Block Name NON_INV(ACA02)
db          79h, 2ch      ;INSAMP_2_NON_INV_CRO
db          7ah, a1h      ;INSAMP_2_NON_INV_CR1
db          7bh, 20h      ;INSAMP_2_NON_INV_CR2
; Instance name PWM16_1, User Module PWM16
;   Instance name PWM16_1, Block Name PWM16_LSB(DCA04)
db          30h, 01h      ;PWM16_1_FUNC_LSB_REG
db          31h, c4h      ;PWM16_1_INPUT_LSB_REG
db          32h, 00h      ;PWM16_1_OUTPUT_LSB_REG
;   Instance name PWM16_1, Block Name PWM16_MSB(DCA05)
db          34h, 21h      ;PWM16_1_FUNC_MSB_REG
db          35h, 34h      ;PWM16_1_INPUT_MSB_REG
db          36h, 05h      ;PWM16_1_OUTPUT_MSB_REG
; Instance name UART_1, User Module UART
;   Instance name UART_1, Block Name RX(DCA07)
db          3ch, 05h      ;UART_1_RX_FUNC_REG
db          3dh, e1h      ;UART_1_RX_INPUT_REG
db          3eh, 00h      ;UART_1_RX_OUTPUT_REG
;   Instance name UART_1, Block Name TX(DCA06)
db          38h, 0dh      ;UART_1_TX_FUNC_REG
db          39h, 01h      ;UART_1_TX_INPUT_REG
db          3ah, 07h      ;UART_1_TX_OUTPUT_REG
db          ffh

```

**FIGURE 7B**

LoadConfigTBL\_project\_Bank0:

; Global Register values

db	60h, 14h	;AnalogColumnInputSelect register
db	63h, 05h	;AnalogReferenceControl register
db	65h, 00h	;AnalogSyncControl register
db	e6h, 00h	;DecimatorControl register
db	02h, 00h	;Port_0_Bypass register
db	06h, f1h	;Port_1_Bypass register
db	0ah, 00h	;Port_2_Bypass register
db	0eh, 00h	;Port_3_Bypass register
db	12h, 00h	;Port_4_Bypass register
db	16h, 00h	;Port_5_Bypass register

; Instance name ADCINC12\_1, User Module ADCINC12

; Instance name ADCINC12\_1, Block Name ADC(ASB20)

; Instance name ADCINC12\_1, Block Name CNT(DBA01)

**FIGURE 7B (Continued)**

```

        db                27h, 00h            ;ADCINC12_1_CounterCR0
        db                25h, 00h            ;ADCINC12_1_CounterDR1
        db                26h, 00h            ;ADCINC12_1_CounterDR2
;   Instance name ADCINC12_1, Block Name TMR(DBA00)
        db                23h, 00h            ;ADCINC12_1_TimerCR0
        db                21h, 00h            ;ADCINC12_1_TimerDR1
        db                22h, 00h            ;ADCINC12_1_TimerDR2
; Instance name Counter16_1, User Module Counter16
;   Instance name Counter16, Block Name CNTR16_LSB(DBA02)
        db                2bh, 00h            ;Counter16_1_CONTROL_LSB_REG
        db                29h, 80h            ;Counter16_1_PERIOD_LSB_REG
        db                2ah, 64h            ;Counter16_1_COMPARE_LSB_REG
;   Instance name Counter16_1, Block Name CNTR16_MSB(DBA03)
        db                2fh, 00h            ;Counter16_1_CONTROL_MSB_REG
        db                2dh, 00h            ;Counter16_1_PERIOD_MSB_REG
        db                2eh, 00h            ;Counter16_1_COMPARE_MSB_REG
; Instance name DAC8_1, User Module DAC8
;   Instance name DAC8_1, Block Name LSB(ASB11)
;   Instance name DAC8_1, Block Name MSB(ASA21)
; Instance name INSAMP_1, User Module INSAMP
;   Instance name INSAMP_1, Block Name INV(ACA01)
;   Instance name INSAMP_1, Block Name NON_INV(ACA00)
; Instance name INSAMP_2, User Module INSAMP
;   Instance name INSAMP_2, Block Name INV(ACA03)
;   Instance name INSAMP_2, Block Name NON_INV(ACA02)
; Instance name PWM16_1, User Module PWM16
;   Instance name PWM16_1, Block Name PWM16_LSB(DCA04)
        db                33h, 00h            ;PWM16_1_CONTROL_LSB_REG
        db                31h, 37h            ;PWM16_1_PERIOD_LSB_REG
        db                32h, 64h            ;PWM16_1_PWIDTH_LSB_REG
;   Instance name PWM16_1, Block Name PWM16_MSB(DCA05)
        db                37h, 00h            ;PWM16_1_CONTROL_MSB_REG
        db                35h, 00h            ;PWM16_1_PERIOD_MSB_REG
        db                36h, 00h            ;PWM16_1_PWIDTH_MSB_REG
; Instance name UART_1, User Module UART
;   Instance name UART_1, Block Name RX(DCA07)
        db                3fh, 00h            ;UART_1_RX_CONTROL_REG
        db                3dh, 00h            ;UART_1_
        db                3eh, 00h            ;UART_1_RX_BUFFER_REG
;   Instance name UART_1, Block Name TX(DCA06)
        db                3bh, 00h            ;UART_1_TX_CONTROL_REG
        db                39h, 00h            ;UART_1_TX_BUFFER_REG
        db                3ah, 00h            ;UART_1_
        db                ffh

```

;Configuration file trailer Config.asm

**FIGURE 7C**

```

; Config.asm
;
; This file is generated by the Device Editor on Application Generation.
; It contains code which loads the configuration data table generated in
; the file ConfigTBL.asm
;

export LoadConfigInit
export _LoadConfigInit
Export LoadConfig_project
export _LoadConfig_project

Flag_CFG_MASK:      equ    10h          ;M8C flag register REG address bit
mask
END_CONFIG_TABLE:   equ    ffh          ;end of config table indicator

_LoadConfigInit:
LoadConfigInit:
    lcall    LoadConfig_project

    ret

;
;Load Configuration project
;
_LoadConfig_project:
LoadConfig_project:
    or          F, FLAG_CFG_MASK          ;set for
bank 1
    mov         A, >LoadConfigTBL_project_Bank1 ;load bank 1 table
    mov         X, <LoadConfigTBL_project_Bank1
    call        LoadConfig                ;load the
bank 1 values
    and         F, ~FLAG_CFG_MASK          ;switch
to bank 0
    mov         A, >LoadConfigTBL_project_Bank0 ;load bank 0 table
    mov         X, <LoadConfigTBL_project_Bank0
    call        LoadConfig                ;load the
bank 0 values
    ret

;
; LoadConfig
;
; This function is not exported. It assumes that the addresses of the table to be loaded
; is contained in the X and A registers as if a romx instruction is the next instruction

```

## FIGURE 8A

```

; to be executed, i.e. lower address in X and upper address in A. There is no return value.
;
LoadConfig:
LoadConfigLp:
    push    X                                ;save config table address on stack
    push    A
    romx                                ;load config address
    cmp     A, END_CONFIG_TABLE           ;check for end of table
    jz      EndLoadConfig                 ;if so, end of load
    mov     X, SP                         ;save the address away
    mov     [X], A
    pop     A                             ;retrieve the table address
    pop     X
    inc     X                             ;advance to the data byte
    jnc     NoOverflow1                   ;check for overflow
    inc     A                             ;if so, increment MSB
NoOverflow1:
    PUSH    x                             ;save the config table address
again:
    push    A
    romx                                ;load the config data
    mov     X, SP                         ;retrieve the config address
    mov     X, [X]
    mov     reg[X], A                    ;write the config data
    pop     A                             ;retrieve the table address
    pop     X
    inc     X                             ;advance to the next
address:
    jnc     NoOverflow2                   ;check for overflow
    inc     A                             ;if so, increment MSB
NoOverflow2:
    jmp     LoadConfigLp                ;loop back
EndLoadConfig:
    pop     A                             ;clean up the stack
    pop     A
    ret

```

FIGURE 8B

```

.. *****
.. *****
..
;; ADCINC12.asm
..
..
;; Assembler source for the 12 bit Incremental
;; A/D converter.
..
.. *****
.. *****

```

```

export ADCINC12_1_Start
export _ADCINC12_1_Start
export ADCINC12_1_SetPower
export _ADCINC12_1_SetPower
export ADCINC12_1_Stop
export _ADCINC12_1_Stop
export ADCINC12_1_GetSamples
export _ADCINC12_1_GetSamples
export ADCINC12_1_StopAD
export _ADCINC12_1_StopAD
export ADCINC12_1_flgdata
export _ADCINC12_1_flgdata
export ADCINC12_1_GetData
export _ADCINC12_1_GetData
export ADCINC12_1_ClearFlag
export _ADCINC12_1_ClearFlag

```

```

include "ADCINC12_1.inc"
include "m8c.inc"

```

```

LowByte: equ 1
HighByte: equ 0

```

```

;; -----
;; Start:
;; SetPower:
;; Applies power setting to the module's analog
;; Programmable System block
;; INPUTS: A contains the power setting
;; OUTPUTS: None.
;; -----

```

```

ADCINC12_1_Start:
_ADCINC12_1_Start:
ADCINC12_1_SetPower:
_ADCINC12_1_SetPower:
    and A,03h
    or A,f0h

```

24/34

```

mov reg[ADCINC12_1_AtoDcr3],A
ret

```

```

;; -----
;; Stop:
;; SetPower:
;; Removes power setting to the module's analog
;; Programmable System block
;; INPUTS: None
;; OUTPUTS: None.
;; -----

```

```

ADCINC12_1_Stop:
_ADCINC12_1_Stop:
    and reg[ADCINC12_1_AtoDcr3],~03h
    ret

```

```

;; -----
;; Get_Samples:
;; SetPower:
;; Starts the A/D convertor and will place data in
;;memory. A flag
;; is set whenever a new data value is available.
;; INPUTS: A passes the number of samples (0
;;is continuous).
;; OUTPUTS: None.
;; -----

```

```

ADCINC12_1_GetSamples:
_ADCINC12_1_GetSamples:
    mov [ADCINC12_1_blnCrC],A    ;number
;of samples
    or reg[INT_MSK1],(ADCINC12_1_TimerMask |
ADCINC12_1_CounterMask)
                                ;enable both interrupts
    Mov[ADCINC12_1_cTimerU],0    ;Force the
;Timer to do one cycle of rest
    or reg[ADCINC12_1_AtoDcr3],10h ;force the
;Integrator into reset
    mov[ADCINC12_1_cCounterU],ffh ;Initialize
;Counter

```

```

    mov[ADCINC12_1_TimerDR1],ffh
    mov[ADCINC12_1_CounterDR1],ffh
    mov[ADCINC12_1_TimerCR0],01h ;enable
;the Timer
    mov[ADCINC12_1_flgcr],00h    ;A/D Data
;Ready Flag is reset
    ret

```

FIGURE 9A

```

;; -----
;; StopAD:
;; Completely shuts down the A/D in an orderly
;; manner. Both the
;; Timer and Counter Interrupts are disabled.
;; INPUTS: None.
;; OUTPUTS: None.
;; -----
ADCINC12_1_StopAD:
  _ADCINC12_1_StopAD:
    mov[ADCINC12_1_TimerCR0],00h
;;disable the Timer
    mov[ADCINC12_1_CounterCR0],00h
;;disable the Counter
    nop
    nop
    and
reg[INT_MSK1],~(ADCINC12_1_TimerMask |
ADCINC12_1_CounterMask)
                                ;Disable both
;;interrupts
    or reg[ADCINC12_1_AtoDc3],10h      ;reset
;;Intergrator
    ret
;; -----
;; flddata:
;; Returns the status of the A/D Data
;; is set whenever a new data value is available.
;; INPUTS: None.
;; OUTPUTS: A returned data status A=:0 no
;;data available
;;          !=: 0 data available.
;; -----
ADCINC12_1_flldata:
  _ADCINC12_1_flldata:
    movA,[ADCINC12_1_flgcr]
    ret
;; -----
;; iGetData:
;; Returns the status from the A/D. Does not
;;check if data is
;; available.
;; is set whenever a new data value is available.
;; INPUTS: None.
;; OUTPUTS: X:A returns the A/D data value.
;; -----

```

```

ADCINC12_1_iGetData:
  _ADCINC12_1_iGet Data:
    mov X,[(ADCINC12_1_flgcr+HighByte)]
    mov A,[(ADCINC12_1_flgcr+LowByte)]
    ret

```

```

;; -----
;; ClearFlag:
;; clears the data ready flag.
;; INPUTS: None
;; OUTPUTS: None.
;; -----
ADCINC12_1_ClearFlag:
  _ADCINC12_1_ClearFlag:
    mov [ADCINC12_1_flgcr],00h
    ret

```

```

ADCINC12_1_API_End:

```

FIGURE 9B



```

HEADER FILES// *****
// *****
//
// ADCINC12_1.h for the 12 bit incremental A/D converter
//
// C declarations for the ADCINC12 User Module
//
//
// *****
//*****

#define ADCINC12_1_OFF    0
#define ADCINC12_1_LOWPOWER    1
#define ADCINC12_1_MEDPOWER    2
#define ADCINC12_1_HIGHPOWER    3

#pragma fastcall ADCINC12_1_Start
#pragma fastcall ADCINC12_1_SetPower
#pragma fastcall ADCINC12_1_GetSamples
#pragma fastcall ADCINC12_1_StopAD
#pragma fastcall ADCINC12_1_Stop

#pragma fastcall ADCINC12_1_flsData
#pragma fastcall ADCINC12_1_iGetData
#pragma fastcall ADCINC12_1_ClearFlag

extern void ADCINC12_1_Start(char power);
extern void ADCINC12_1_SetPower(char power);
extern void ADCINC12_1_GetSamples(char chout);
extern void ADCINC12_1_StopAD(void);
extern void ADCINC12_1_Stop(void);

extern char ADCINC12_1_flsData(void);
extern int ADCINC12_1_iGetData(void);
extern void ADCINC12_1_ClearFlag(void);

```

FIGURE 10

```

.. *****
..
..
..
;; ADCINC12_1.inc for the 12 bit incremental A/D converter
..
..
;; Assembler declarations for the ADCINC12 User Module
....
;;;
.. *****
..
.. *****
..

ADCINC12_1_AtoDcr0:    equ    90h
ADCINC12_1_AtoDcr1:    equ    91h
ADCINC12_1_AtoDcr2:    equ    92h
ADCINC12_1_AtoDcr3:    equ    93h
ADCINC12_1_CounterFN:   equ    24h
ADCINC12_1_CounterSL:   equ    25h
ADCINC12_1_CounterOS:   equ    26h
ADCINC12_1_CounterDR0:  equ    24h
ADCINC12_1_CounterDR1:  equ    25h
ADCINC12_1_CounterDR2:  equ    26h
ADCINC12_1_CounterCR0:  equ    27h
ADCINC12_1_TimerFN:     equ    20h
ADCINC12_1_TimerSL:     equ    21h
ADCINC12_1_TimerOS:     equ    22h
ADCINC12_1_TimerDR0:    equ    20h
ADCINC12_1_TimerDR1:    equ    21h
ADCINC12_1_TimerDR2:    equ    22h
ADCINC12_1_TimerCR0:    equ    23h
ADCINC12_1_TimerMask:   equ    01h
ADCINC12_1_CounterMask: equ    02h
ADCINC12_1_OFF:         equ    0
ADCINC12_1_LOWPOWER:    equ    1
ADCINC12_1_MEDPOWER:    equ    2
ADCINC12_1_HIGHPOWER:   equ    3
ADCINC12_1_NUMBITS:     equ    12

```

FIGURE 11

```

.. *****
;;
;; ADCINC12int.asm
;;
;;
;; Assembler source for interrupt routines the 12 bit Incremental
;; A/D converter.
;;
.. *****
;;

export ADCINC12_1_CNT_INT
export _ADCINC12_1_TMR_INT
include "ADCINC12_1.inc"
include "m8c.inc"

Area bss(RAM)
    ADCINC12_1_cTimerU: BLK 1 ;The upper byte of the Timer
    ADCINC12_1_cCounterU: BLK 1 ;The Upper byte of the Counter
    ADCINC12_1_ilncr:
    ADCINC12_1_ilncr: BLK 2 ;A/D value
    ADCINC12_1_flncr
    ADCINC12_1_flncr: BLK 1 ;Data Valid Flag
    ADCINC12_1_blncrC: BLK 1 ;# of times to run A/D

area text(ROM,REL)

export _ADCINC12_1_cTimerU
export _ADCINC12_1_cCounterU
export _ADCINC12_1_ilncr
export _ADCINC12_1_ilncr
export _ADCINC12_1_flncr
export _ADCINC12_1_flncr
export _ADCINC12_1_blncrC

LowByte: equ 1
HighByte: equ 0

;; -----
;; CNT_INT:
;; Decrement the upper (software) half on the counter whenever the
;; lower (hardware) half of the counter underflows.
;; INPUTS: None.
;; OUTPUTS: None.
;; -----
ADCINC12_1_CNT_INT:
    dec[_ADCINC12_1_cCounterU]
    reti

```

FIGURE 12A

```

;; -----
;; TMR_INT:
;; This routine allows the counter to collect data for 64 timer cycles
;; This routine then holds the integrator in reset for one cycle while
;; the A/D value is calculated.
;; INPUTS: None.
;; OUTPUTS: None.
;; -----
ADCINC12_1_TMR_INT:
    dec[ADCINC12_1_cTimerU]
; if(upper count >= 0)
    jc else 1
    reti
else 1: (upper count decremented pass 0)
    tst reg{ADCINC_1_AtoDcr3}, 10h ;to change when ice is fixed dbz
    jz else2

```

FIGURE 12A (Continued)

```

; if(A/D has been in reset mode)
    mov reg[ADCINC12_1_CounterCR0],01h    ;Enable Counter
    and reg[ADCINC12_1_AtoDcr3],~10h      ;Enable Analog Counter
    mov reg[ADCINC12_1_cTimerU],((1<<(ADCINC12_1_NUMBITS-6))-1)
                                           ;This will be the real counter value
    reti
else2:;(A/D has been in integrate mode)
    mov reg[ADCINC12_1_CounterCR0],00h    ;disable Counter
    or F,01h                              ;Enable the interrupts
;~~~~~
; Good place to add code to switch inputs for multiplexed input to ADC
;~~~~~
    or reg[ADCINC12_1_AtoDcr3],10h ;Reset Integrator
    mov [(ADCINC12_1_ilncr+LowByte)],ffh
    mov [(ADCINC12_1_ilncr+HighByte)],(ffh-(ADCINC12_1_NUMBITS-7))
;
    push A
    mov A, reg[ADCINC12_1_CounterDR0],01h ;read Counter
    mov A, reg[ADCINC12_1_CounterDR2],01h ;now you really read the data
    sub[(ADCINC12_1_ilncr+LowByte)],A
    mov A, reg[ADCINC12_1_cCounterU]
    sbb[(ADCINC12_1_ilncr+HighByte)],A
    pop A
    cmp[(ADCINC12_1_ilncr+HighByte)],(1<<(ADCINC12_1_NUMBITS-7))
    jnz endif10
; if(max positive value)
    dec[(ADCINC12_1_ilncr+HighByte)]
    mov[(ADCINC12_1_ilncr+LowByte)],ffh
endif10:
    asr[(ADCINC12_1_ilncr+HighByte)]      ;divide by 4
    rrc[(ADCINC12_1_ilncr+LowByte)]
    asr[(ADCINC12_1_ilncr+HighByte)]
    rrc[(ADCINC12_1_ilncr+LowByte)]

    mov[ADCINC12_1_flgcr],01h             ;set AD data flag

```

FIGURE 12B

```

.....
; User code here for interrupt system.
.....
    cmp[ADCINC12_1_blnrC],00h
    jz endif3
; if(ADCINC12_1_blnrC is not zero)
    dec[ADCINC12_1_blnrC]
    jnz endif4
; if(ADCINC12_1_blnrC has decremented down to zero to 0))
    mov reg[ADCINC12_1_TimerCR0],00h        ;disable the Timer
    mov reg[ADCINC12_1_CounterCR0],00h      ;disable the Counter
    nop
    nop
    and reg[INT_MSK1],~(ADCINC12_1_TimerMask | ADCINC12_1_CounterMask)
                                           ;disable both interrupts
    or reg[ADCINC12_1_AtoDcr3],10h          ;reset Integrator
    reti
endif4;;
endif3;;
endif2;;
    mov [ADCINC12_1_cTimerU],00h            ;Set Timer for one cycle of reset
    mov [ADCINC12_1_cCounterU],ffh         ;Set Counter hardware for easy enable
    mov reg[ADCINC12_1_CounterDR1],ffh
    reti
endif1;;

```

ADCINC12\_1\_APIINT\_End: A/D converter

FIGURE 12B (Continued)

1300

```

;-----
; Interrupt Vector Table
;-----
;
; Interrupt vector table entries are 4 bytes long
; and contain the code
; that services the interrupt (or causes it to be
; serviced).
;-----

Area TOP(ROM,ABS)

org 0          ;Reset Interrupr Vector
jmp_start      ;First Instruction
; executed following a Reset

org 04h        ;Supply Monitor Interrupt
; Vector
//call void_handler
reti

org08h         ;Block DBA00
; Interrupt Vector
ljmp ADCINC12_1_TMR_INT
reti

org0Ch         ;Block DBA01
; Interrupt Vector
ljmp ADCINC12_1_CNT_INT
reti

org10h        ;Block DBA02
; Interrupt Vector
//call void_handler
reti

org14h        ;Block DBA03
; Interrupt Vector
ljmp Counter16_1INT
reti

org 18h       ;Block DCA04
; Interrupt Vector
//call void_handler
reti

```

32/34

```

org 1Ch          ;Block DCA0
; Interrupt Vector
ljmp PWM16_1INT
reti

org 20h          ;Block DCA06
; Interrupt Vector
ljmp UART_1TX_INT
reti

org 24h          ;Block DCA07
; Interrupt Vector
ljmp UART_1RX_INT
reti

org 28h          ;Analog Column 0
; Interrupt Vector
//call void_handler
reti

org 2Ch          ;Analog Column 1
; Interrupt Vector
//call void_handler
reti

org 30h          ;Analog Column 2
; Interrupt Vector
//call void_handler
reti

org 34h          ;Analog Column 3
; Interrupt Vector
//call void_handler
reti

org 38h          ;GPIO Interrupt Vector
//call void_handler
reti

org 3Ch          ;Sleep Timer Interrupt
; Vector
jmp SleepTimerISR
reti

```

FIGURE 13A

1351 boot.asm Interrupt Name	1352 Data Sheet Interrupt Name	1353 Type
Start	Reset	Fixed
Interrupt1	Supply Monitor	Fixed
Interrupt2	DBA00	Programmable System Block
Interrupt3	DBA01	Programmable System Block
Interrupt4	DBA02	Programmable System Block
Interrupt5	DBA03	Programmable System Block
Interrupt6	DCA04	Programmable System Block
Interrupt7	DCA05	Programmable System Block
Interrupt8	DCA06	Programmable System Block
Interrupt9	DCA07	Programmable System Block
Interrupt10	Analog Column 0	Programmable System Block
Interrupt11	Analog Column 1	Programmable System Block
Interrupt12	Analog Column 2	Programmable System Block
Interrupt13	Analog Column 3	Programmable System Block
Interrupt14	GPIO	Fixed
Interrupt15	SLEEP TIMER	Fixed

FIGURE 13B



100

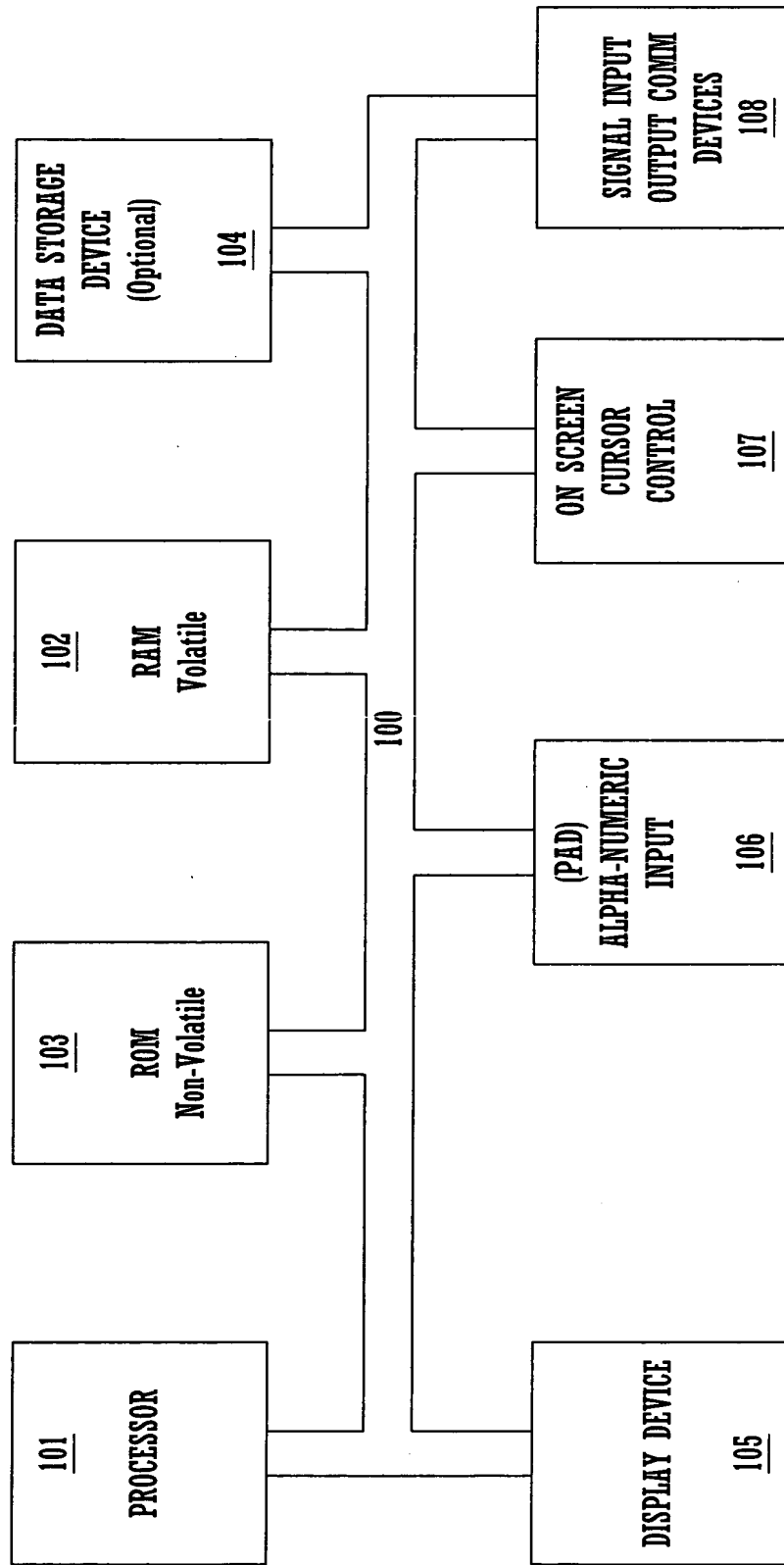


FIGURE 14